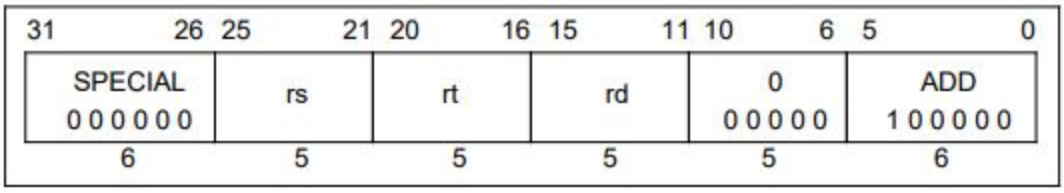
# ADD — Add — ADD




**Format:**

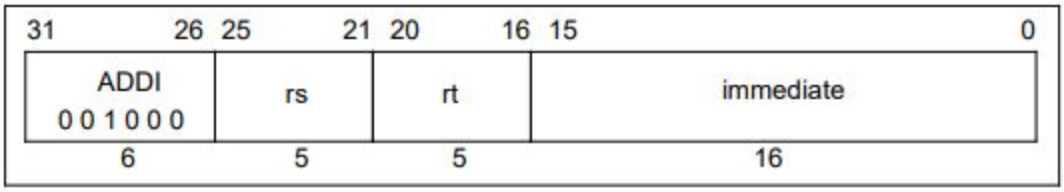
ADD rd, rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*. In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

An overflow exception occurs if the carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

# ADDI — Add Immediate — ADDI




**Format:**

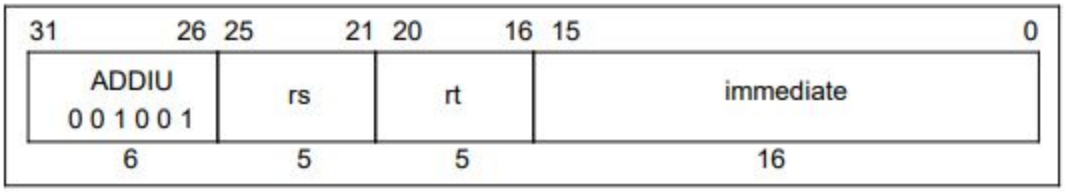
ADDI rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt*. In 64-bit mode, the operand must be valid sign-extended, 32-bit values.

An overflow exception occurs if carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rt* is not modified when an integer overflow exception occurs.

# ADDIU Add Immediate Unsigned ADDIU




**Format:**

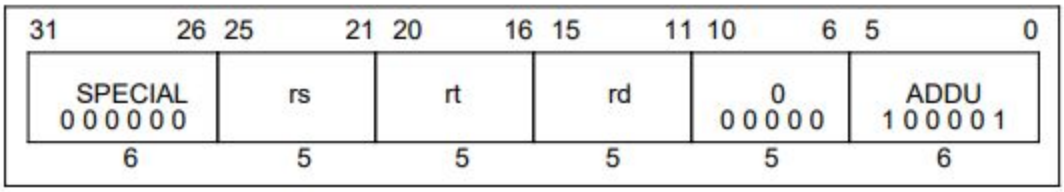
ADDIU rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt*. No integer overflow exception occurs under any circumstances. In 64-bit mode, the operand must be valid sign-extended, 32-bit values.

The only difference between this instruction and the ADDI instruction is that ADDIU never causes an overflow exception.

# ADDU Add Unsigned ADDU




**Format:**

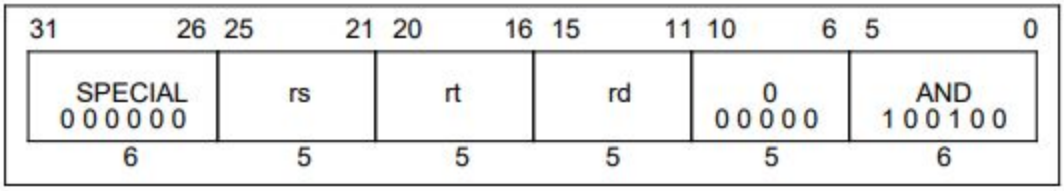
ADDU rd, rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*. No overflow exception occurs under any circumstances. In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the ADD instruction is that ADDU never causes an overflow exception.

# AND — And — AND



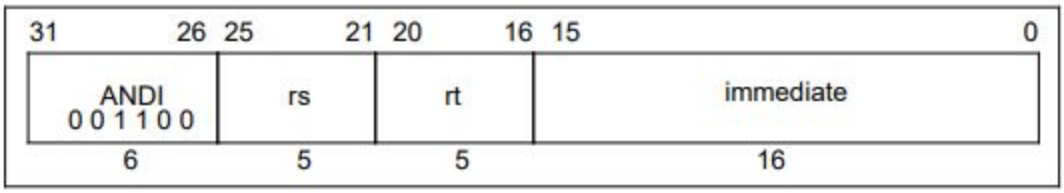

**Format:**

AND rd, rs, rt

**Description:**

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical AND operation. The result is placed into general register *rd*.

# ANDI — And Immediate — ANDI



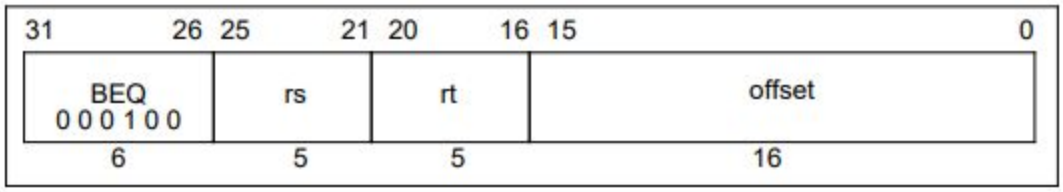

**Format:**

ANDI rt, rs, immediate

**Description:**

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical AND operation. The result is placed into general register *rt*.

# BEQ — Branch On Equal — BEQ



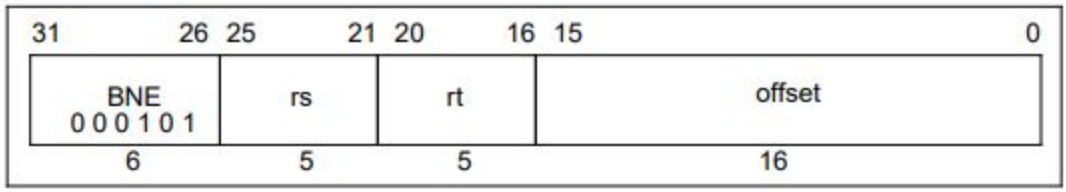

**Format:**

BEQ rs, rt, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register ***rs*** and the contents of general register ***rt*** are compared. If the two registers are equal, then the program branches to the target address, with a delay of one instruction.

# BNE — Branch On Not Equal — BNE



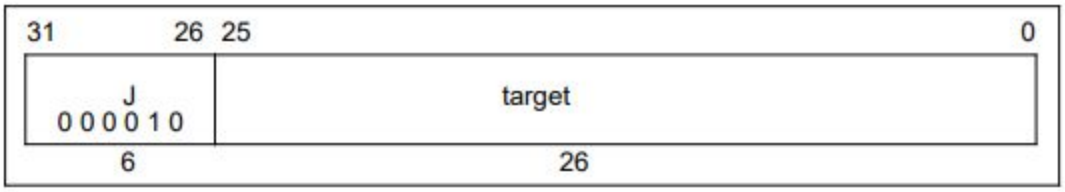

**Format:**

BNE rs, rt, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register ***rs*** and the contents of general register ***rt*** are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

# J — Jump — J



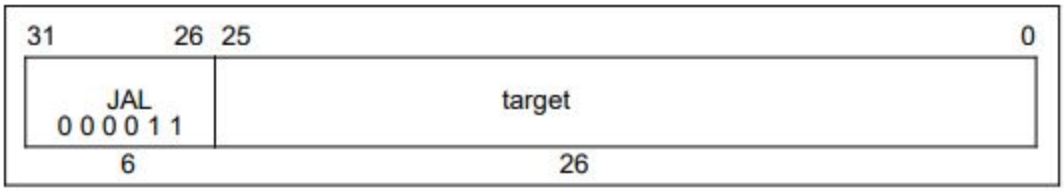

**Format:**

J target

**Description:**

The 26-bit target address is shifted left two bits and combined with the high-order bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction.

# JAL — Jump And Link — JAL



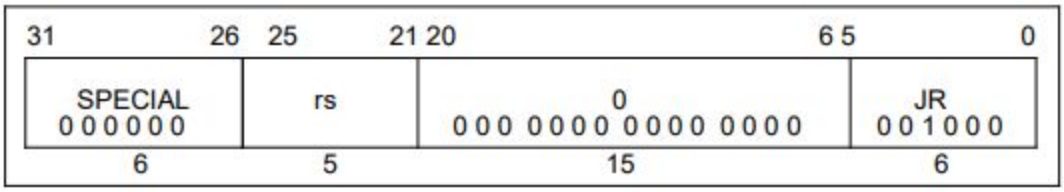

**Format:**

JAL target

**Description:**

The 26-bit target address is shifted left two bits and combined with the high-order bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction after the delay slot is placed in the link register, ***r31.***

**JR** **Jump Register** **JR**




**Format:**

JR rs

**Description:**

The program unconditionally jumps to the address contained in general register *rs*, with a delay of one instruction.

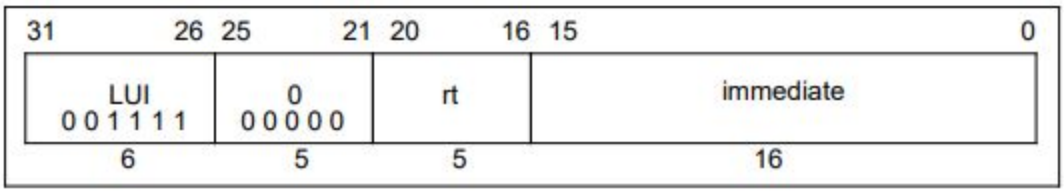
Since instructions must be word-aligned, a **Jump Register** instruction must specify a target register (*rs*) whose two low-order bits are zero. If these low-order bits are not zero, an address exception will occur when the jump target instruction is subsequently fetched.

# LA

La is a pseudo instruction and should be converted to one or two assembly instructions.

- For example: la $2, VAR1 : VAR1 is a label in the data section
- It should be converted to lui and ori instructions.

**LUI** **Load Upper Immediate** **LUI**



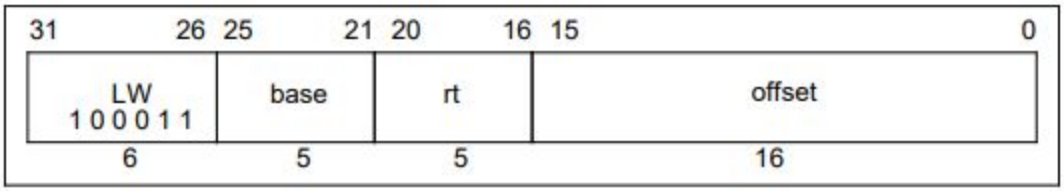

**Format:**

LUI rt, immediate

**Description:**

The 16-bit *immediate* is shifted left 16 bits and concatenated to 16 bits of zeros. The result is placed into general register *rt*. In 64-bit mode, the loaded word is sign-extended.

# LW Load Word LW



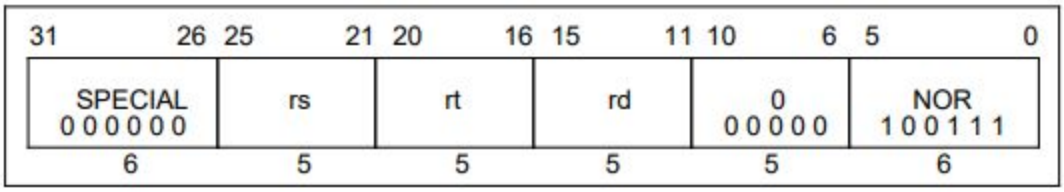

**Format:**

LW rt, offset(base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the word at the memory location specified by the effective address are loaded into general register *rt*. In 64-bit mode, the loaded word is sign-extended. If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

# NOR Nor NOR



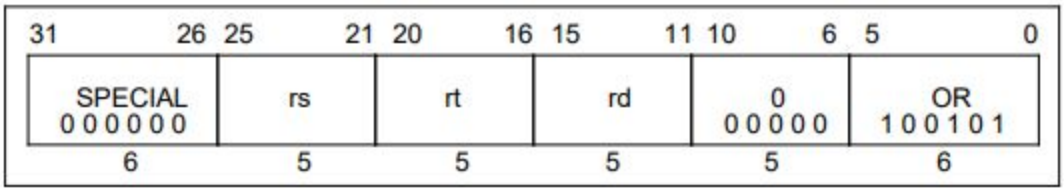

**Format:**

NOR rd, rs, rt

**Description:**

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical NOR operation. The result is placed into general register *rd*.

# OR Or OR



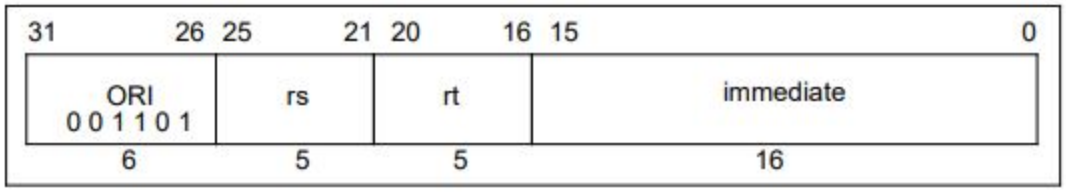

**Format:**

OR rd, rs, rt

**Description:**

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical OR operation. The result is placed into general register *rd*.

# ORI Or Immediate ORI



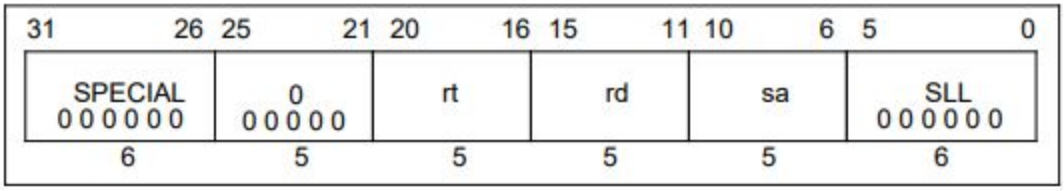

**Format:**

ORI rt, rs, immediate

**Description:**

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical OR operation. The result is placed into general register *rt*.

# SLL — Shift Left Logical — SLL




**Format:**

SLL rd, rt, sa

**Description:**

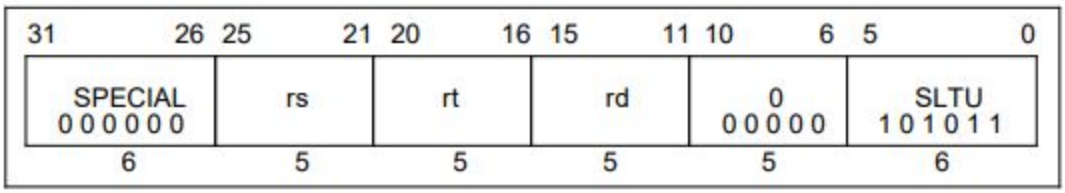
The contents of general register *rt* are shifted left by *sa* bits, inserting zeros into the low-order bits.

The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign extended when placed in the destination register. It is sign extended for all shift amounts, including zero; SLL with a zero shift amount truncates a 64-bit value to 32 bits and then sign extends this 32-bit value. SLL, unlike nearly all other word operations, does not require an operand to be a properly sign-extended word value to produce a valid sign-extended word result.

**NOTE:** SLL with a shift amount of zero may be treated as a NOP by some assemblers, at some optimization levels. If using SLL with a zero shift to truncate 64-bit values, check the assembler you are using.

# SLTU — Set On Less Than Unsigned — SLTU




**Format:**

SLTU rd, rs, rt

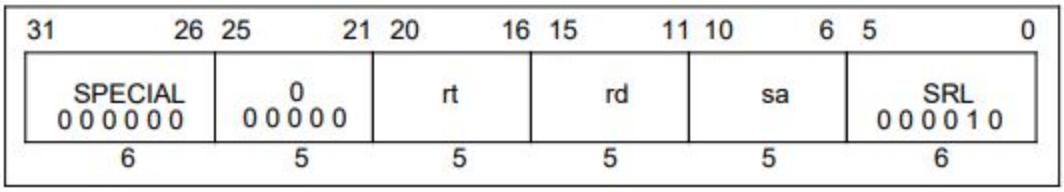
**Description:**

The contents of general register *rt* are subtracted from the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to one; otherwise the result is set to zero.

The result is placed into general register *rd*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

# SRL — Shift Right Logical — SRL




**Format:**

SRL rd, rt, sa

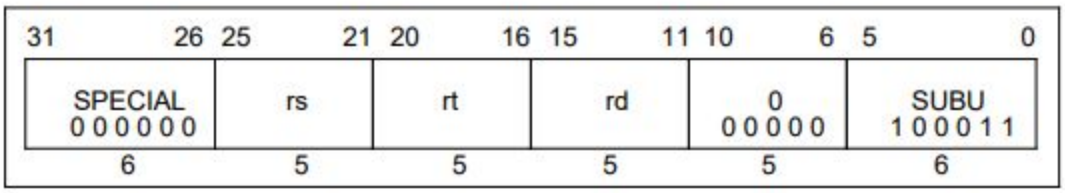
**Description:**

The contents of general register *rt* are shifted right by *sa* bits, inserting zeros into the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

# SUBU — Subtract Unsigned — SUBU




**Format:**

SUBU rd, rs, rt

**Description:**

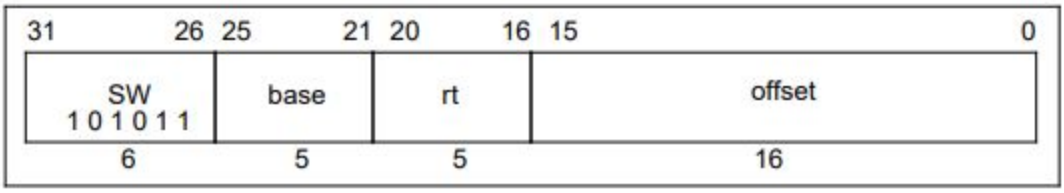
The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result.

The result is placed into general register *rd*.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the SUB instruction is that SUBU never traps on overflow. No integer overflow exception occurs under any circumstances.

# SW — Store Word — SW




**Format:**

SW rt, offset(base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of general register *rt* are stored at the memory location specified by the effective address.

If either of the two least-significant bits of the effective address are non-zero, an address error exception occurs.